An Efficient Data Selection Policy for Search Engine Cache Management

Xinhua Dong, Ruixuan Li, Heng He, Xiwu Gu
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, P.R.China
Email: {xhdong, rxli, henghe, guxiwu}@hust.edu.cn

Mudar Sarem
School of Software Engineering
Huazhong University of Science and Technology
Wuhan, P.R.China
Email: mudar66@hotmail.com

Meikang Qiu
Department of Computer Science
Pace University
New York, USA
Email: mqiu@pace.edu

Keqin Li
Department of Computer Science
State University of New York
New York, USA
Email: lik@newpaltz.edu

***Abstract*—Caching is an effective optimization in search engine. The data selection policy plays a key role in caching, which places the data to be cached in memory. However, the current data selection policies are not suitable to the hybrid storage architecture with solid state disks (SSDs), which have gradually replaced hard disk drives (HDDs) in search engines. In this paper, we present an Efficient Data Selection policy (EDS) for search engine cache management, which views cache media as a knapsack, and views results and posting lists as items. The best benefit can be computed by greedy algorithms. In order to verify the effectiveness, we carry out a series of experiments to study essential factors of data selection in different architectures, including HDD, SSD, and SSD-based hybrid storage architecture, which uses SSD as a secondary cache for memory. The experimental results demonstrate that the proposed policy improves the hit ratio by 20.04% and the retrieval performance on HDD, SSD, and hybrid architecture by 31.98%, 28.72% and 23.24%, respectively.**

***Index Terms*—search engine; cache management; data selection; solid state disk; hybrid storage architecture**

I. Introduction

In a modern search engine, caching is the preferred technique for attaining performance. Over the past years, many caching techniques have been developed and used in search engines. In order to reduce the query response time, the search engines commonly dedicated portions of the servers memory to cache certain query results [1], posting list [2][3][4], intersection [5], document [6], score, and snippet [7]. Caching means copying frequently or recently accessed parts of the data from high-capacity but slow storage devices (HDD) to low-capacity but fast storage devices (Memory or SDD). The data selection means selecting the effective data to be placed in memory or SSD. These caches avoid excessive disk access and repeated computation.

Due to the wide speed gap between the random read and the sequential read in HDD, the benefit of the cache hit has been largely attributed to the saving of the expensive random read operations. Therefore, the early studies focused on the improvement of the hit ratio in the HDD search engine architecture, and put forward some classic policies [3][4], such as Freq, FreqSize etc. The emerging SSD has ultra-high performance for random data access. Random reads in SSD are one to two orders of magnitude faster than in HDD [8]. In an SSD-based search engine infrastructure, now the benefit of the cache hit should attribute to both the saving of the random read and the saving of the subsequent reads. Since SSD is replacing HDD, the cache hit ratio is no longer a reliable reflection of the actual query latency because a larger data items being found in the cache yields a higher query latency improvement over a smaller data item [8]. The frequency was found to be a core factor in the SSD experiments. In an SSD-based hybrid search engine infrastructure, our previous proposed CBSLRU [9] algorithm gained higher hit ratios than the traditional LRU.

However, without being guided by a particular theory, these proposed factors may be not comprehensive, and some combinations of the factors that influence the retrieval performance have not been found. The proposed policies mainly consider the frequency and the size, but many other factors have not been considered. Therefore, the applications may not work well in different hardware environments. We assume that the historical query log has a guiding role for the future query [1]. Therefore, a set of query features could be found through query log. When the user submits a query, the data selection policy is tuned to use these features. In order to analyze the query log and improve the efficiency of the data selection, we have introduced the knapsack problem, which views cache media as knapsack and views result and posting list as items, and used a greedy algorithm to calculate the retrieval time. Also, we propose an effective data selection policy (EDS), which involves the frequency, the size, the disk parameters, and the other factors. The EDS can be better applied to the different storage architectures.

We have made three main contributions in our work. First, we describe and analyze the knapsack problem in different storage architectures. Second, through derivation and comparison, we find some critical factors that affect the performance of the search engines. Third, we propose data selection policy (EDS) that places the efficient data to be cached either in

memory or SSD.

The rest of the paper is organized as follows. In Section II, we discuss the related work. Section III presents different storage architectures for search engines. In Section IV, we analyze the knapsack problem and perform the derivation of the EDS in hybrid storage architectures. Section V shows the results of the performance evaluation. Finally, in Section VI, we conclude this study and discuss the future work.

II. Related Work

In general, search engine caches can be classified into two categories: static caches [2] and dynamic caches [10]. The static caches try to capture the access locality of the data items. Past data access logs are utilized to determine the data that should be cached. Typically, the items that are more frequently accessed in the past are preferred over the infrequently accessed items for caching. Static caches need to be periodically updated, depending on the variation of the access frequencies of the items. On the other hand, the dynamic caches try to capture the recency of the data access. The data that is more likely to be accessed in the near future remains in the cache. The challenge in the previous research in dynamic caching was to develop cache eviction policies [11]. Recently, the current challenge is to devise effective policies to retain cache.

The caching techniques in search engines can be classified into two-level caching and multi-level caching. Saraiva et al. [12] evaluated a two-level caching architecture using result and list caching on the search engine TodoBR. Result caching filtered out the repetition in the query stream by caching the complete results of the previous queries for a limited time window [10][11]. Long et al. [13] proposed and evaluated a three-level caching scheme that added an intermediate level of caching. On this basis, Wang et al. [8] also carried out a series of experiments on document and snippet in the proposed web search engine architecture. Ozcan et al. [5] proposed five-level static cache architecture for web search engines.

Considering the special I/O performance of the SSD, the researches on such SSD-based hybrid storage architecture have been attracting both academic and industrial fields. With its excellent random read performance, the SSD can work well as a read cache in front of a larger HDD [14]. Suk et al. [15] proposed a hybrid file system, called hybridFS, whose primary objective is to put together attractive features of both the HDD and the SSD devices, to construct a large-scale, virtualized address space with a minimum cost.

III. Storage Architectures of Search Engines

In this section, we briefly present the storage architecture of a search engine, including one-level and two-level architectures. Then, we give the data management process of the two-level search engine architecture.

A. One-level architecture

The normal search engine employs one-level architecture. Memory cache is used to store all of the intermediate results includes result cache, posting list cache, snippet cache, and document cache. The global inverted indexes are stored on the HDD or SSD. That is to say, there are two situations in one-level architecture, one can be called "Memory + HDD" and the other is called "Memory + SSD".

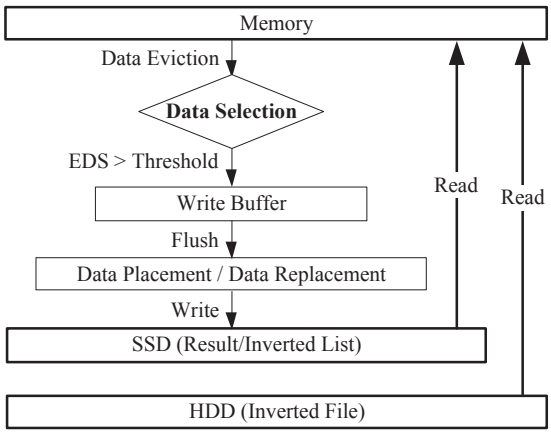

Fig. 1. The data management process of two-level search engine architecture

B. Two-level architecture

The cache storage structure can be divided into two levels: level 1 cache (L1 cache) and level 2 cache (L2 cache). L1 cache refers to the memory, and its capacity is usually several GB to dozens of GB. L2 cache refers to the SSD, and its capacity is usually dozens of GB to hundreds of GB. In this paper, we adopt the hybrid scheme, which the data in memory may or may not be cached on SSD, depending on a criteria either set by the user or decided based on the current workload. Figure 1 shows the data management process of the two-level search engine architecture. According to the data flow, the main steps are as follows.

First, when the user submits a query, Lucene will read the data from the HDD (i. e., the inverted lists), and put it in memory. Second, once the memory buffer overflows, the Lucene will eliminate some of the data according to the cache replacement strategy. The data will be eliminated by the filter of a data selection module. By screening ($EDS > Threshold$), the data will be added to the write buffer (called Write Buffer). Third, when the Write Buffer overflows, the Lucene will brush the data of the Writer Buffer into the SSD according to either the data placement strategy or the data replacement strategy. Fourth, when the data in the SSD being hit, the Lucene will read the corresponding data from the SSD to the memory.

IV. Effective Data Selection in Storage Architectures

In this section, we first describe knapsack problem in the SSD-based hybrid storage architecture, and then we define a variety of time cost in the query process. Finally, we give detailed derivation steps of an effective data selection (EDS) in different storage architectures.

TABLE I
RETRIEVAL UNDER DIFFERENT SITUATIONS IN HYBRID ARCHITECTURE

Situation	Memory	SSD	HDD	Probability	Time Cost
S_1	R			P_1	$T_1 = C_{mpr}$
S_2		R		P_2	$T_2 = C_{spr}$
S_3	I			P_3	$T_3 = C_{mpl} + C_0$
S_4		I		P_4	$T_4 = C_{spl} + C_0$
S_5			I	P_5	$T_5 = C_{hpl} + C_0$

A. Knapsack problem in hybrid architecture

During the retrieval process based on our proposed hybrid storage architecture, there are five kinds of basic situations of data access, which are shown in Table I.

In order to decrease the amount of memory space used in hybrid architecture, only a part of the results are permitted to be stored on SSD. From Table I, We note that Memory and HDD denotes where the data read from, R denotes the results, and I denotes the inverted lists. Probability represents the probability that the corresponding situation takes place. Time is the average retrieval time accordingly. In the calculation formula of the time cost, C_{mpr} represents the time cost of obtaining search results directly from Memory, and C_{spr} represents the time cost of obtaining the search results directly from SSD. C_{mpl} represents the time cost of obtaining inverted lists from Memory, and C_{hpl} represents the time cost of obtaining inverted lists from HDD. So, $C_0 = C_{rank} + C_{doc} + C_{snip}$, This previous formula represents the sum of the time cost of obtaining the search results by sorting, obtaining documents, and generating snippet. It is easy to know that $T_5 \geq T_k (1 \leq k \leq 4)$. From Table I, we give the average retrieval time as shown in the following Formula 1.

$$AVG(T) = \sum_{i=1}^{5} T_i \times P_i \tag{1}$$

In order to minimize the average retrieval time, we need to make full use of the advantages of the hybrid cache by analyzing the different situations in Table I, which means that the probability of "S_1", "S_2", "S_3", and "S_4" should be increased. Therefore, we analyze the time cost and the frequency of the single item, which include result and inverted list. Suppose that there are *n* results and *m* inverted lists. Each item will be accessed with a certain frequency (f). By considering that an item may either be cached in Memory or SSD, or be obtained from HDD indirectly, Formula 2 can then represent the mathematical expectations of the retrieval.

$$\begin{aligned} &E(T) = \\ &\sum_{i=1}^{n} (T_1 \cdot x_{iMM} + T_2 \cdot x_{iSSD} + T_5 \cdot (1 - x_{iMM} - x_{iSSD})) \cdot f_i \\ &+ \sum_{j=1}^{m} (T_3 \cdot x_{jMM} + T_4 \cdot x_{jSSD} + T_5 \cdot (1 - x_{jMM} - x_{jSSD})) \cdot f_j \\ &= \sum_{i=1}^{n} T_5 \cdot f_i - \sum_{i=1}^{n} [(T_5 - T_1) \cdot f_i \cdot x_{iMM} + (T_5 - T_2) \cdot f_i \cdot x_{iSSD}] \\ &+ \sum_{j=1}^{m} T_5 \cdot f_j - \sum_{j=1}^{m} [(T_5 - T_3) \cdot f_j \cdot x_{jMM} + (T_5 - T_4) \cdot f_j \cdot x_{jSSD}] \end{aligned} \tag{2}$$

$x_{i(store)}$ = 1 represents that item *i* is cached in the storage medium store, otherwise, $x_{i(store)}$ = 0. For example, x_{iMM} = 1 represents that item *i* is cached in main memory. For the purpose of minimizing the average retrieval time, we want the mathematical expectations of the access time as small as possible, and then we get the following Formula 3.

$$\begin{aligned} &minAVG(T) \Leftrightarrow minE(T) \Leftrightarrow \\ &max \sum_{i=1}^{n} [(T_5 - T_1) \cdot f_i \cdot x_{iMM} + (T_5 - T_2) \cdot f_i \cdot x_{iSSD}] \\ &+ \sum_{j=1}^{m} [(T_5 - T_3) \cdot f_j \cdot x_{jMM} + (T_5 - T_4) \cdot f_j \cdot x_{jSSD}] \end{aligned} \tag{3}$$

In the objective function, we can take $(T_5 - T_k) f_i$ as the value of the item, while it means the time savings by the cache. Therefore, it can be converted into a multi-knapsack problem. By considering the different cache spaces of the Memory and the SSD, we build a mathematical model of a multi-knapsack problem for our analysis. The items which will be loaded into the knapsacks including *n* search results and *m* inverted lists. Each item has its own size (W_i), access frequency (f_i), and retrieval time (as shown in Table I). Also, the items need to meet the limitations of the memory capacity C_{MM} and the SSD capacity C_{SSD}. There is no intersection between the set of items cached in memory and SSD. According to above goals and these condition constraints, the multi-knapsack problem can be described as a mathematical model presented in the following Formula 4.

$$\begin{aligned} &max \sum_{i=1}^{n} [(T_5 - T_1) \cdot f_i \cdot x_{iMM} + (T_5 - T_2) \cdot f_i \cdot x_{iSSD}] \\ &+ \sum_{j=1}^{m} [(T_5 - T_3) \cdot f_j \cdot x_{jMM} + (T_5 - T_4) \cdot f_j \cdot x_{jSSD}] \\ &s.t. \sum_{i=1}^{n} W_i \cdot x_{iMM} + \sum_{j=1}^{m} W_j \cdot x_{jMM} \leq C_{MM} \\ &\sum_{i=1}^{n} W_i \cdot x_{iSSD} + \sum_{j=1}^{m} W_j \cdot x_{jSSD} \leq C_{SSD} \\ &x_{iMM} + x_{iSSD} = 0 \; or \; 1 \qquad (1 \leq i \leq n) \\ &x_{jMM} + x_{jSSD} = 0 \; or \; 1 \qquad (1 \leq j \leq m) \\ &x_{iMM} = 0 \; or \; 1, x_{iSSD} = 0 \; or \; 1 \qquad (1 \leq i \leq n) \\ &x_{jMM} = 0 \; or \; 1, x_{jSSD} = 0 \; or \; 1 \qquad (1 \leq j \leq m) \end{aligned} \tag{4}$$

As it is known, for the unbounded knapsack problem, the greedy algorithm can achieve the optimal solution. In this model, the size of a single item is far less than the cache capacity of Memory and SSD. Therefore, for saving the computing cost, we use the greedy algorithm to solve the knapsack problem. Similarly, we sort the items in ascending order according to the value per unit (UV), which is shown in the following Formula 5.

$$UV = \frac{(T_5 - T_k) f_i}{W_i} \qquad (1 \leq k \leq 4) \tag{5}$$

B. Definitions of saving time terms

In order to explore the change rule, the time cost associated with each query step is computed using the formulas shown in Table II. As it can be seen from this table, D_{seek} and $D_{rotation}$ represent the seek latency and the rotational delay of the HDD respectively. D_{read} represents the time cost of obtaining one

TABLE II
COST COMPUTATIONS IN THE CACHE OF SEARCH ENGINE

Notation	Computation
C_{hpl}	$D_{seek} + D_{rotation} + D_{read} \times \mid I_j \mid \times S_p \div D_{block}$
C_{spl}	$S_{read} \times \mid I_j \mid \times S_p \div S_{block}$
C_{rank}	$CPU_{scoring} \times \sum_{ti \in q}(\mid I_j \mid \times S_p)$
C_{doc}	$D_{seek} + D_{rotation} + D_{read} \times \mid d_{top} \mid \div D_{block}$
C_{Sdoc}	$S_{read} \times \mid d_{top} \mid \div S_{block}$
C_{snip}	$CPU_{snippet} \times \mid d \mid$
C_{spr}	$S_{read} \times \mid R_{top} \mid \times S_r \div S_{block}$
C_{mpl}	$M_{read} \times \mid I_j \mid \times S_p \div D_{block}$
C_{mpr}	$M_{read} \times \mid R_{top} \mid \times S_r \div S_{block}$

TABLE III
THE TIME SAVING WITH EACH TYPE OF KNAPSACK

Notation	Type	Time saving
Saving1	$T_5 - T_1$	$C_{hpl} + C_0 - C_{mpr} \approx C_{hpl} + C_0$
Saving2	$T_5 - T_2$	$C_{hpl} + C_0 - C_{spr} \approx C_{hpl} + C_0$
Saving3	$T_5 - T_3$	$C_{hpl} - C_{mpl} \approx C_{hpl}$
Saving4	$T_5 - T_4$	$C_{hpl} - C_{spl} \approx C_{hpl}$

block of data from the HDD, as S_{read} and M_{read} do from the SSD and the Memory respectively. D_{block} is the block size of the HDD, and S_{block} is the block size of the SSD. I_i represents the number of DocId of the *i*th posting list, and S_p is the size of the storage of per DocId. $CPU_{scoring}$ and $CPU_{snippet}$ represent the scoring cost and the snippet generation cost respectively. d_{top} represents the size of the highest scoring document, and *d* is the number of documents. R_{top} represents the highest scoring result. Finally, S_r is the average size of per result.

C. Derivation of EDS in storage architectures

Compared with the inverted list entries, the result entries are quite small and similar in size, so we can take common policy to deal with the results. At the same time, we need some special selection policies for the inverted list.

Based on Formula 5,we can find that the following three factors are associated with the value of UV: saving time ($T_5 - T_k$), access frequency (f_i) and item size (W_i). We set EDS as the effective values per unit, and put the larger ones into the knapsack as presented in the following Formula 6:

$$EDS = \frac{Saving \times f_i}{W_i} \tag{6}$$

Based on different storage architecture, the EDS can be analyzed from three aspects. First, according to the time-consuming listed in Table I, we find that there are greater differences in time saving between the items cached in memory and those cached in SSD. Therefore, we can divide the multi-knapsack problem into two stages, the memory knapsack problem and the SSD knapsack problem. Considering the differences in the time saving, and the size between the results and the inverted lists, we separate these two types of items, and get two knapsack problems. In this way, the multi-knapsack problem is transformed into four basic 0-1 knapsack problems. The time saving with each query step is deduced using the formulas shown in the following Table III.

By calculating the parameters of the hardware and analyzing the query log, we find that the time consumption of C_{mpr}, C_{mpl}, C_{spr} and C_{spl} can be negligible relative to the time of reading the same result and inverted list from the HDD. Through observation, we find that C_{hpl} is the key factor to the saving time of reading the inverted lists, and $C_{hpl} + C_0$ is the key factor to the saving time of reading the result. By substituting C_{hpl} and $(C_{hpl} + C_0)$ into Formula 6, we can obtain Formula 7 and 8.

$$\begin{aligned} EDS_{PL} &= \frac{Freq \times C_{hpl}}{W_j} \\ &= C_1 \frac{Freq}{W_j} + C_2 Freq \qquad (C_1 = D_{seek} + D_{rotation}, C_2 = \frac{D_{read}}{D_{block}}) \end{aligned} \tag{7}$$

$$\begin{aligned} EDS_R &= \frac{Freq}{W_i} \times (C_{hpl} + C_0) \approx \frac{Freq}{W_i} \times (C_{hpl} + C_{doc}) \\ &= (2C_1 + C_3) \times \frac{Freq}{W_i} + C_2 \times \frac{W_j}{W_i} \times Freq \qquad (C_3 = d \times K \times C_2) \end{aligned} \tag{8}$$

Second, in similar way, we complete the same steps presented in Sec. IV, part A. In contrast, the object is converted from hybrid storage to SSD. Also, we can use the model of 0-1 knapsack problem and the value per unit (presented in Formula 5). We can obtain the following two formulas 9 and 10, respectively. Where, EDS_{SPL} represents EDS of the inverted lists and EDS_{SR} represents EDS of the results in SSD.

$$\begin{aligned} EDS_{SPL} &= \frac{Freq \times C_{spl}}{W_j} \\ &= \frac{S_{read}}{S_{block}} \times Freq = C_4 Freq \qquad (C_4 = \frac{S_{read}}{S_{block}}) \end{aligned} \tag{9}$$

$$\begin{aligned} EDS_{SR} &= \frac{Freq}{W_i} \times (C_{spl} + C_0) \approx \frac{Freq}{W_i} \times (C_{spl} + C_{Sdoc}) \\ &= C_5 \times \frac{Freq}{W_i} + C_4 \times \frac{W_j}{W_i} \times Freq \qquad (C_5 = d \times K \times C_4) \end{aligned} \tag{10}$$

Third, when the object is converted from SSD to HDD, we can also obtain the previous two Formula 7 and 8.

As seen in Formula 7, we can compute the value of C_1 and C_2 according to the parameters of certain types of hard disk. (For example, D_{seek} = 8.5ms, $D_{rotation}$= 4.2ms, D_{read} = 4.88ms, S_p = 8bytes, D_{block} = 512 bytes, then C_1 = 12. 7, C_2 = 0. 0095). In Formula 8, we find that the time consumption of C_{rank} and C_{snip} can be negligible relative to the time of C_{doc}. Therefore, the value of C_0 is C_{doc}. K represents the average size per document. When the size of the result is the same (W is a constant), EDS_R is a function of Freq. (For example, *d*= 10, K=8kb, W=4kb, then C_3 =780. 8, EDS_R = 0. 22Freq).

V. PERFORMANCE EVALUATION

In this section, our evaluation includes two aspects, one is to compare the hit ratios of several proposed algorithms, and the other is to verify that our proposed policy can improve the retrieval performance of the search engines, including the HDD architecture, the SSD architecture, and the SSD-based hybrid storage architecture. We preserve the notations of the previous sections here.

TABLE IV
THE EXPERIMENT PLATFORM SPECIFICATIONS

Test-platform Environment	
IR Tool	Lucene 3.5.0
Data Set	enwiki-20090805-pages-articles.xml
Query Log	AOL-user-ct-collection
SSD	Samsung SSD 840 Series 120GB
HDD	Seagate ST2000DM001-1CH164 2TB
OS	Windows 7/Ubuntu 12.04
CPU/RAM	Inter(R) Xeon (R) CPU E3 1230 V2/16G

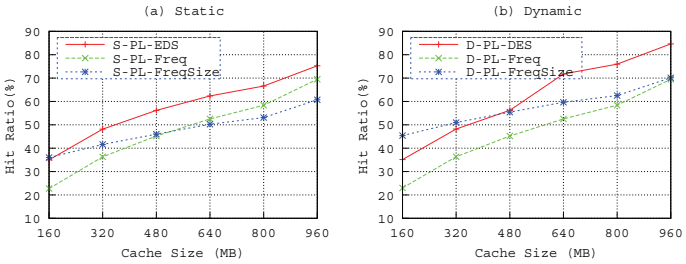

Fig. 2. The hit ratio comparison in HDD

A. Experimental settings

Table IV summarizes the experiment platform specifications. Five million documents indexed from enwiki data set have been used in our experiment, and the used query log was from AOL. In our experiments, we have prepared a set of sample queries from the AOL query log, which were performed the index retrieval. Our simulative search engine is based on Lucene 3.5.0.

B. Hit Ratio

There are two existing static query inverted list caching policies. We refer them as S-PL-Freq and S-PL-FreqSize, respectively. In addition, we refer to the proposed static caching policy as S-PL-EDS. Figure 2 shows the hit ratio comparison with HDD.

In experiments, the number of total documents is 1,000,000, and the cache size ranges from about 160MB to 960MB. Figure 2(a) shows the hit ratio comparison between S-PL-Freq, S-PL-FreqSize and S-PL-EDS in the static case. It can be seen from Figure 2(a) that the hit ratio will increase with the increase of the cache capacity at a certain range, and S-PL-FreqSize, which tends to cache popular terms with short posting lists, has a higher cache hit ratio than S-PL-Freq in the previous stage. However, if we increase the cache capacity continuously, the hit ratio of S-PL-Freq exceeds the hit ratio of S-PL-FreqSize at a later stage. In the static case, S-PL-EDS has the highest cache hit ratio among the three policies.

S-PL-Freq, S-PL-FreqSize and S-PL-EDS has discussed above, for their dynamic cache versions, we refer them as D-PL-Freq, D-PL-FreqSize and D-PL-EDS, respectively. The dynamic cache versions are to capture the recency of the data access. Figure 2(b) shows the hit ratio of D-PL-Freq, D-PL-FreqSize, and D-PL-EDS in the dynamic case. As it can be seen from this figure, D-PL-FreqSize always has a higher hit ratio than D-PL-Freq. The hit ratio of D-PL-FreqSize is too higher than the hit ratio of D-PL-EDS in the previous stage. However, at a later stage, the hit ratio of D-PL-EDS exceeds the hit ratio of D-PL-FreqSize. The reason is that the advantages of caching popular terms have been gradually disappearing. We can compare the changes in the hit ratio of the three policies between the dynamic and the static cases, respectively. Our proposed D-PL-EDS policy improves the hit ratio by 20.04% in average compared with the D-PL-Freq and D-PL-FreqSize policies.

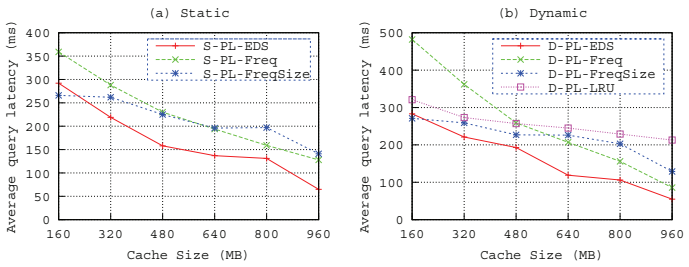

Fig. 3. Performance comparison in HDD

C. Retrieval performance on HDD

Figure 3 presents performance comparison in HDD. Figure 3(a) shows the average query time in the static case. The query latency of the three selected caching policies is consistent with the tradition holds: when one policy has a higher cache hit ratio than the others, its query latency is also shorter than the others. On the whole, the S-PL-EDS policy has the best query latency compared with the S-PL-FreqSize and S-PL-Freq policies. Specifically, we can note that S-PL-EDS's average query time is slightly worse than the average query time of S-PL-FreqSize at 160MB cache memory because the latter has much higher cache hit ratio. Figure 3(b) shows the average query time in the dynamic case. Compared with the static case, the average query latency of the dynamic case has the same trends. In comparison with LRU, the average response latency is reduced by 31.98% in EDS.

D. Retrieval performance on SSD

Figure 4 shows the average query time on SSD in the dynamic case, and the average query time have followed the same trend in the static case. According to Formula 9, EDS_{SPL} only related to the frequency, so D-PL-EDS is the same as D-PL-Freq. However, as it can be seen in figure 4, D-PL-FreqSize becomes poor in terms of query latency. The reason is that the benefit brought by the higher hit ratio of D-PL-FreqSize is watered down by the fewer sequential read savings caused by short posting lists. In comparison with FreqSize, the average response latency is reduced by 28.72% in EDS.

E. Retrieval performance on hybrid architecture

In two-level architecture, the data selection method is relatively complicated. In the dynamic case, the data selection strategy can be implemented when the data flow from the memory to the SSD. If the data is selected, we can use three

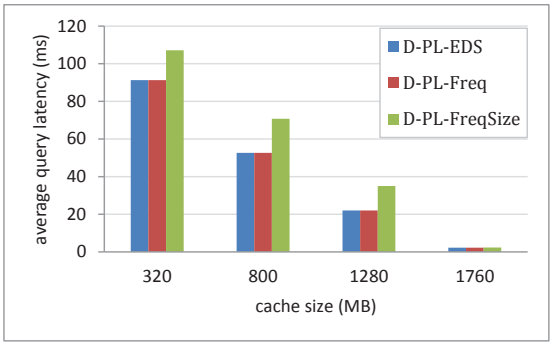

Fig. 4. Performance comparison in SSD

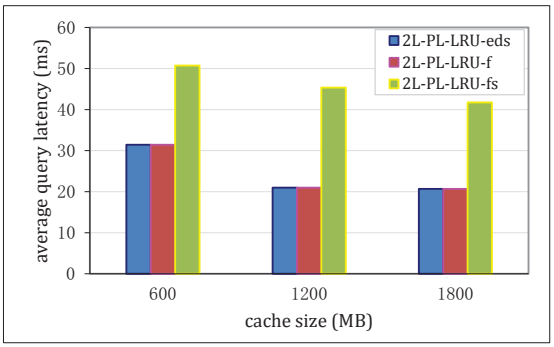

Fig. 5. Performance comparison in hybrid architecture

strategies EDS, Freq and FreqSize. We refer them as 2L-PL-LRU-eds, 2L-PL-LRU-f and 2L-PL-LRU-fs, respectively. As shown in figure 5. By considering that the selected data will be written to the SSD, our proposed EDS method only adopt frequency factor here. Figure 5 shows that the average query time of 2L-PL-LRU-eds (i.e., 2L-PL-LRU-f) is significantly better than that of 2L-PL-LRU-fs. The reason is that the cache terms with high frequency save query time.In this experiment, the average response latency is reduced by 23.24% in EDS compared to FreqSize.

The experimental results have shown that our proposed EDS policy is better than the Freq and the FreqSize policies in the performance for different architectures. We believe that there are two main reasons for this achievement. First, the EDS policy has the highest cache hit ratio. Second, the EDS policy can be viewed as a reasonable compromise between the FreqSize policy and the Freq policy. The value of EDS depends on the parameters of the specific storage medium. Through the guidance of the query log and the EDS policy, the ideal data are always placed in the cache.

VI. Conclusion and Future Work

In this paper, we describe three types of storage architecture for search engines, and then we define and analyze knapsack problem in storage architectures. Through derivation and comparison, we find some critical factors, which affect the performance of search engines. Then, we propose a data selection policy (EDS), which places the efficient data to be cached in memory or SSD. The experimental results demonstrate our proposed EDS policy. In the future, we will consider the data selection policies of intersection cache and document cache. In addition, we need to reduce the cost of search engine servers without influencing the retrieval performance.

Acknowledgments

This work is supported by National Natural Science Foundation of China under grants 61173170, 61300222, 61433006 and U1401258, Innovation Fund of Huazhong University of Science and Technology under grants 2015TS069 and 2013QN120, and Science and Technology Support Program of Hubei Province under grant 2014BCH270. Meikang Qiu is supported by NSF 1457506.

References

[1] R. Ozcan, I. S. Altingovde, and U. Ulusoy, “Static query result caching revisited,” Proc. of the 17th International Conference on World Wide Web (WWW’08), Beijing, China, pp.1169-1170, 2008

[2] R. A. Baeza-Yates and F. Saint-Jean, “A three level search engine index based in query log distribution,” Proc. of the 10th International Symposium on String Processing and Information Retrieval (SPIRE’03), Manaus, Brazil, pp.56-65, 2003

[3] R.A. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras and F. Silvestri, “The impact of caching on search engines,” Proc. of the 30th International conference on research and development in Information Retrieval (SIGIR’07), Amsterdam, Netherlands, pp.183-190, 2007

[4] R.A. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras and F. Silvestri, “Design trade-offs for search engine caching,” ACM Trans. Web, vol.2(4), pp.1-28, 2008

[5] R. Ozcan, I. S. Altingovde, B. B. Cambazoglu, F. P. Junqueira and U. Ulusoy, “A five-level static cache architecture for web search engines,” Inf. Process. Manage. vol.48(5), pp.828-840, 2011

[6] A. Turpin, Y. Tsegay, D. Hawking and H. E. Williams, “Fast generation of result snippets in web search,” Proc. of the 30th International conference on research and development in Information Retrieval (SIGIR’07), Amsterdam, Netherlands, pp.127-134, 2007

[7] D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego and F. Silvestri, “Caching query-biased snippets for efficient retrieval,” Proc. of the 14th International Conference on Extending Database Technology (EDBT’11), Uppsala, Sweden, pp.93-104, 2011

[8] J.G. Wang, E. Lo, M. L. Yiu, J.C. Tong, G. Wang and X.G. Liu, “The impact of solid state drive on search engine cache management,” Proc. of the 36th International conference on research and development in Information Retrieval (SIGIR’13), Dublin, Ireland , pp.693-702, 2013

[9] R. X. Li, C. Z. Li, W. J. Xiao, H. Jin, H. He, X.W. Gu, K. M. Wen, Z. Y. Xu, “An Efficient SSD-based Hybrid Storage Architecture for Large-Scale Search Engines,” Proc. of the 41st International Conference on Parallel Processing (ICPP’12), Pittsburgh, PA, USA, pp.450-459, 2012

[10] T. Fagni, R. Perego, F. Silvestri, S. Orlando, “Boosting the performance of Web search engines: Caching and prefetching query results by exploiting historical usage data.” Trans. Inf. Syst. vol.24(1), pp.51-78, 2006

[11] Q. Gan and T. Suel, “Improved techniques for result caching in web search engines,” Proc. of the 18th International Conference on World Wide Web (WWW’09), Madrid, Spain, pp.431-440, 2009

[12] P.C. Saraiva, E.S. de Moura, R.C. Fonseca, Jr. W. Meira, B.A. Ribeiro-Neto and N. Ziviani, “Rank-Preserving Two-Level Caching for Scalable Search Engines,” Proc. of the 24th International conference on research and development in Information Retrieval (SIGIR’01), New Orleans, Louisiana, USA, pp.51-58, 2001

[13] X. Long and T. Suel, “Three-Level Caching for Efficient Query Processing in Large Web Search Engines,” Proc. of the 14th international conference on World Wide Web (WWW’05), Chiba, Japan, pp.369-395, 2005

[14] J. Matthews, S.N. Trika, D. Hensgen, R. Coulson, K. Grimsrud, “Intel Turbo Memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems,” TOS. vol.4(2), pp.1-24, 2008

[15] J. Suk, J. No and Y.K. Kim, “Design and implementation of hybridFS,” Proc. of the 3rd International Conference on Computer Science and Information Technology (CSAIT’10), Amsterdam, Netherlands, pp.501-505, 2010